Lab 10

Get checked off for up to 3 points of incomplete work from the previous lab within the first 10 minutes of the lab.
For lab 10, you will not be checked off for 3 points of incomplete work next week. So try to finish the lab in the given lab time

In this lab, you can form a group of 2-3 individuals. You must be checked off together as a group at the end of the lab. Although you perform tasks as a group, ensure that you understand the work and ask questions to TAs as needed.

Problem statement:
Towers of Hanoi is a simple puzzle that we're going to be using as practice for recursion and 2D arrays. The puzzle itself is very simple– it consists of three columns arranged from left to right, and some number of disks N of different sizes. To begin, the N disks are placed on the 1st column in order of their size, with the largest disk at the bottom of the column. The puzzle's goal is to finish with the disks arranged in the same order (biggest on the bottom, smallest on the top) on the 2nd column. Of course, you can't just move the disks however you want! You need to follow these rules:

- You can only move one disk at a time by taking it off the top of its peg and putting it onto another peg.
- You're not allowed to place a disk on top of another disk that's smaller– that is, every disk must be smaller than every disk beneath it on the peg.

You can do the simulation here: https://www.mathsisfun.com/games/towerofhanoi.html

Here is an outline of the recursive towers function, feel free to add another parameter if needed, i.e. **total_disks**.:

```
void towers(int number_of_disks, int b[ ][3], int from_col, int to_col, int spare) {
        If(number of disks is >= 1)
                Call Towers with (number_of_disks-1, b, from_ col, spare, to_ col)
                Move the disk
                Print the board
                Call Towers with (number_of_disks-1, b, spare, to_ col, from_ col)
}
```

(5 pts) Implementation : Dynamically Allocated 2-D array

In the previous lab, we statically allocated 2-Darray. Next, implement this is using a dynamically allocated **2-D array with 3 columns for the 3 posts and N rows for N disks.** Get the number of disks from the user as a command-line argument, i.e. towers 5.

Continue to initialize the array with the numbers corresponding to the disks in the first column and 0s in all other columns to represent the initial state of the game. You should now see the above example output, given 2 for the number of disks.

Remember to change your towers() and print_array() function parameters to accept dynamically allocated arrays, rather than statically allocated. To help you out, your towers() function will be change to the following prototype:

```
void towers(int number_of_disks, int **b, int from_col, int to_col, int spare);
```

Make sure you delete your board after calling the towers function.

Create/Delete Functions for Dynamically Allocated 2-D array

If you haven't done so already, create functions for creating and deleting the array on the heap. Make sure you set the board back to null in the delete function!

Run your program through valgrind to make sure you do not have any memory leaks!!!

(5 pts) Computation

A true magic square is one where the numbers in a 2-d array are arranged such that the sum of each column, each row, and each diagonal equal the same number. Example of a magic square (all columns, rows, and diagonals equal 15!!!):

2 7 6
9 5 1
4 3 8

Write a program to create a 3x3 two-dimensional array on the stack:

- Fill the array with random numbers, 1 – 9 (inclusive), for each element in the array
- Write a function, bool fun(int a[][3], int size), to:
  - Determines if the array is a magic square.
  - Return true if the array is magic, and false otherwise.

Show your completed work and answers to the TAs for credit. You will not get points if you do not get checked off!

Computation Reference Document

1. Pay close attention to these **requirements**:
   a. No global variables.
   b. No memory leaks
   c. No use of help or man.
   d. No use of the internet.
2. **Wait to begin** until informed by the Proctors.
3. You are encouraged to **spend time with design**
4. Begin entering in your code using a **Linux editor**
   a. You may use vi, vim, or emacs as your editor
5. You are also allowed to **compile, test, and debug** your work.
   g++ test.cpp
   ./a.out
6. If you accidentally freeze your screen by typing ctrl+s, use **ctrl+q** to unfreeze it. 7. **When you are finished**, wait for the Proctor to check you off.
   a. Show TAs your design, if any.
   b. Show, compile, and run your program for the proctor.
   c. Remove your test.cpp file

Main Template/Libraries for Common Built-in Functions:

```
/* Ascii values:
48-57 //0-9
65-90 //A-Z
97-122 //a-z
*/
#include <iostream> /* cin, cout, endl */
#include <cmath> /* pow(), sqrt()
#include <string> /* .size(), .length(), .at()
#include <cstdlib> /* srand(), rand(), atoi() */
#include <ctime> /* time() */
#include <cstring> /* cstring type, strlen(), strcmp() */ using namespace

std;

int main (){
 //Example usage of random numbers
 srand (time(NULL)); //seed random generator
 rand()%10; //gives 0-9

 //Example usage of C strings, where s is a cstring
 strlen(s); //gives number of characters in string
 atoi(s); //gives integer value of digit characters

 return 0;
}
```